Ontology of architectural decisions supporting ATAM based assessment of SOA architectures

Piotr Szwed*, Paweł Skrzynski*, Grzegorz Rogus* and Jan Werewka*
*AGH University of Science and Technology
Department of Applied Computer Science
Email: {pszwed,skrzynia,rogus,werewka}@agh.edu.pl

***Abstract*—Nowadays, Service Oriented Architecture (SOA) might be treated as a state of the art approach to the design and implementation of enterprise software. Contemporary software developed according to SOA paradigm is a complex structure, often integrating various platforms, technologies, products and design patterns. Hence, it arises a problem of early evaluation of a software architecture to detect design flaws that might compromise expected system qualities. Such assessment requires extensive knowledge gathering information on various types of architectural decisions, their relations and influences on quality attributes. In this paper we describe SOAROAD (SOA Related Ontology of Architectural Decisions), which was developed to support the evaluation of architectures of information systems using SOA technologies. The main goal of the ontology is to provide constructs for documenting SOA. However, it is designed to support future reasoning about architecture quality and for building a common knowledge base. When building the ontology we focused on the requirements of Architecture Tradeoff Analysis Method (ATAM) which was chosen as a reference methodology of architecture evaluation.**

***Index Terms*—software architecture, ontology, SOA, ATAM, architecture assessment, architecture evaluation, enterprise architecture**

I. Introduction

NOWADAYS, Service Oriented Architecture (SOA) might be treated as a state of the art approach to the design and implementation of enterprise software, which is driven by business requirements. Within the last decade a number of concepts related to SOA have been developed, including ESB (Enterprise Service Bus), web services, design patterns, service orchestration and choreography and various security standards. Due to the fact that there are many technologies that cover the area of SOA, the development and evaluation of SOA compliant architectures is especially interesting.

SOAROAD has been designed as a methodology for the assessment of software architectures developed according to SOA principles. It is based on the Architecture Tradeoff Analysis Method (ATAM) [11], [5], which is a mature, scenario-based, early method for architecture assessment. ATAM defines a quality model, an organizational framework for evaluation process and expected results: sensitivity points, tradeoffs and risks. A limitation of the ATAM method is that it depends on experts knowledge, perception and previous experience. It may easily happen, that an inexperienced evaluator overlooks some implicit decisions and risks introduced by them.

In the SOAROAD approach the very basic set of ATAM terms used to describe architecture is enriched by including common terminology and relationships between concepts related to various aspects of service oriented architecture design and development. The gathered knowledge, formalized as an ontology, facilitates performing an assessment in more exhaustive manner, helping to ask questions, revealing implicit design decisions and obtaining more reliable results.

The contribution of the paper is a proposal of a SOAROAD ontology as a tool supporting scenario based assessment of systems following a service-orientation paradigm and service design, development and deployment.

II. Related works

Architecture evaluation has attracted many researchers and practitioners during the last 20 years. A survey paper on this topic [18] lists 37 methods of architecture evaluation, classifying them according to two dimensions: location in the software lifecycle (early vs. late) and element being analyzed (system architecture, isolated architectural style or a design pattern). The paper suggests that scenario-based methods, including SAAM [12] and ATAM [11], [5] can be considered as a mature, reliable and easy to implement in practical situations. There are several reports on successful applications of ATAM for assessment of a battlefield control system [13], wargame simulation [10], product line architecture [8], control of a transportation system [3], credit card transactions system[16] and a dynamic map system [21]. Recently, a few extensions of ATAM were proposed, including a combination with the Analytical Hierarchy Process [24] and APTIA [14].

The application of ontologies to provide a systematic and formal description of architectural decisions was first proposed by Kruchten in [15]. The ontology distinguished several types of decisions that can be applied to software architecture and its development process. Main categories included: Existence, Ban, Property and Executive decisions. The ontology defined also attributes, which were used to describe decisions, including states (Idea, Tentative, Decided, Rejected, etc.). In [7] an ontology supporting ATAM based evaluation was proposed. The ontology specified concepts covering the ATAM model of architecture, quality attributes, architectural styles and decisions, as well as influence relations between elements of

This work was supported from AGH University of Science and Technology under Grant No. 11.11.120.859

architectural style and quality attributes. The effort to structure the knowledge about architectural decisions, was accompanied by works aimed at a development of tools enabling the edition and graphical visualization of design decisions, often in a collaborative mode, e.g. [4], [6], [17].

III. A CONCEPT OF APPLICATION OF SOAROAD ONTOLOGY

The SOAROAD (SOA Related Ontology for Architectural Decisions) has four main goals, it should: (1) provide a comprehensive description of architectural views, i.e. components and their connections; (2) gather a domain knowledge providing a unified vocabulary related to SOA and enterprise architecture; (3) help to ask question about various properties of architectural design and decisions; (4) be capable of representing assignments of properties relevant to SOA compliant technologies to elements of system architecture.

It was assumed that the ontology would follow a foundational model (ontology skeleton) defining various properties corresponding to design decisions that can be attributed to components, connections, interfaces and compositions. If applicable, these design decisions can be supplemented by additional relations. The ontology would also specify design patterns.

Another assumption is related to a distribution of the knowledge between ontology TBox (set of classes, their attributes and relations) and ABox (individuals, values of their attributes and relationships). The types of elements appearing in architectural views are classified in the TBox. Concrete elements, e.g. those appearing in the diagrams of architectural views, are represented as individuals in an ABox. The ontology describes types of design decisions (properties) as classes, whereas their values as individuals that can be directly assigned to elements of architectural views or linked to form trees.

The concept of the ontology application is presented in the Fig. 1 The process of building an architecture description starts with eliciting *Architecture views ABox*, i.e. a set of linked components, interfaces and connections. This model can be prepared either manually or with the support of dedicated import tools converting ArchiMate [22] models of Archi editor [1] or UML [19], e.g. from VisualParadigm.

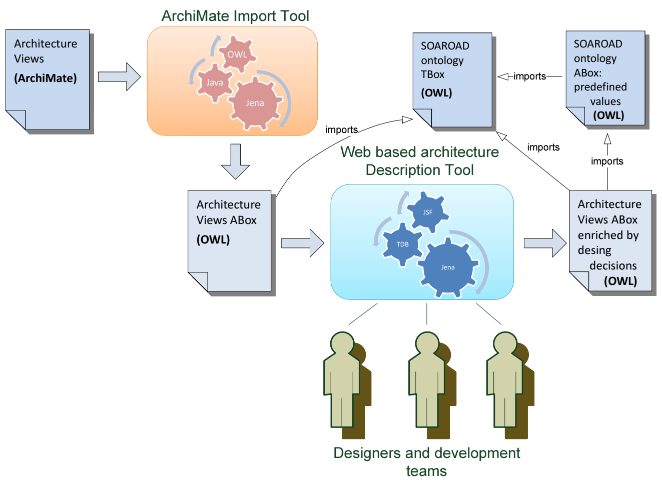
ArchiMate Import Tool
Architecture Views (ArchiMate)
OWL
Java
Jena
SOAROAD ontology TBox (OWL)
imports
SOAROAD ontology ABox: predefined values (OWL)
imports
imports
imports
Web based architecture Description Tool
JSF
TDB
Jena
Architecture Views ABox (OWL)
Architecture Views ABox enriched by desing decisions (OWL)
Designers and development teams

Fig. 1. A concept of application of SOAROAD ontology

A web based tool supporting architecture description uses the classes and individuals defined in the *SOAROAD ontology Domain Description TBox* and *SOAROAD Architectural decisions ABox* to generate forms or questionnaires in which software architects or members of development teams can make assignments of property values to elements of architecture views.

The resulting *Detailed Architecture ABox* refers elements of *Architecture views ABox* and individuals defined in SOAROAD ontology (merging two input ontologies and asserting additional relations). This ontology serves as a detailed architecture documentation within a software development project. It can be examined either manually or with use of automated tools.

IV. ONTOLOGY DESCRIPTION

The SOAROAD ontology was built in three steps. Firstly, a foundational model serving as ontology skeleton was proposed. Then we manually gathered and analyzed information related to service oriented architectures, technologies, architectural approaches, design patterns, etc. originating from various sources: books, technical papers, reference manuals and Internet resources. Finally, the ontology was populated with this information by translating intermediate textual description into OWL constructs. At present the ontology consists of 110 classes, 9 object properties and 105 individuals.

The basic model of software architecture used in ATAM [2] defines it after [20] as a set of components and linking them connections. We extend this simplistic model by defining *Interfaces* and *Functions* of components as presented in Fig. 2. A connection links a component having the caller role with an interface (calee). Components, connections and interfaces can be attributed with: *ComponentProperties*, *ConnectionProperties* and *InterfaceProperties* respectively (Fig. 3). Examples of such properties are: platform, web service type, communication type, queueing and query granularity.

Composition is a coherent set of components and connectors. System architecture is itself a composition. For the purpose of analysis we may focus on a particular subset of components and connectors and describe their properties, e.g. a distribution of queries among several databases building up a composition or realization of a design pattern.

During the ATAM based evaluation the overall system architecture and properties of its parts are analyzed to establish scenario responses and achievements of corresponding quality attributes. It may be, however, observed that some architecture properties or their combinations have known influence on quality attributes, e.g. a use of asynchronous web services or applying MVC design pattern, which increases modifiability and a granularity of queries, has an impact on performance. This kind of knowledge can be expressed via *influences* relations.

Architectural decision is an assignment of a property value to a component, interface, connection or a composition. In this context the terms *property* and *architectural decision* can be used to some extent interchangeably. However, it may happen that certain decisions or components are dependent on previously assigned properties. An example of such a dependency is the composition type – a property assigned to a set (composition) of web service components. Selecting orchestration as the composition type requires that an orchestration component, e.g. BPEL capable module is used. The *required* relation or its subproperties in the ontological model express this dependency.

The assumed foundational model adopts a reification strategy while modeling various properties of an architectural design. Properties are defined as classes, whose individuals can be linked by additional relations indicating specific roles. An example of such a property is MVC design pattern, which requires the identification of components playing the roles of a Model (typically a database), a Controller (e.g. an EJB) and a View (e.g. a set of HTML pages produced by JSP scripts).

For each property, that can be treated as a class of design decision, a number of individuals (corresponding to decision values) is defined. They can be selected in assignments, e.g. *JavaEECompliantAS* (a subclass of *ComponentProperty*) has several predefined individuals: *JBoss*, *Glassfish*, *WebLogic*, *Web-Sphere*, *ColdFusion*, etc.

Example ontology assertions related to component properties are presented in Table I and Table II. A property (an ontology class) is followed by property values (individuals in the ontology) put in parentheses.

Apart from defining design decisions, the ontology specifies functions of components. Class Function contains classes of entities such as: *Routing*, *MessageMapping*, *ProtocolSwitch*, *MediationService*, *MessageValidation*, *AuditFunction*, *DatabaseIntegration*, etc.

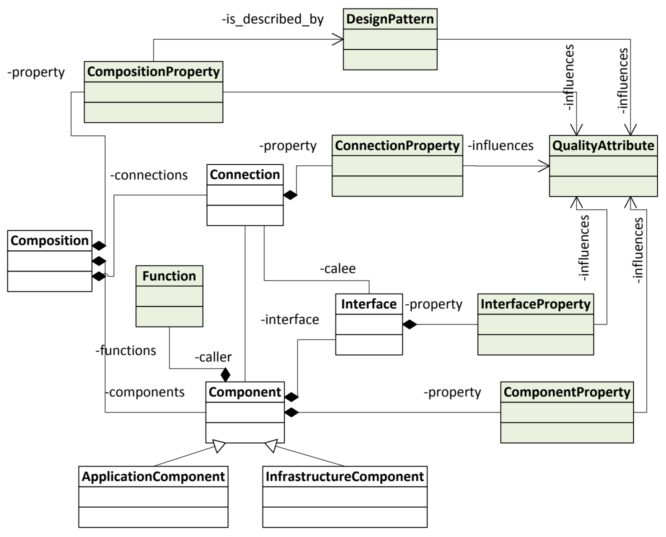

Fig. 2. Foundational model of software architecture and its properties

TABLE I
COMPONENT PROPERTIES

Property (values)	Description
PlatformTechnology (CORBA, EJB, JINI, RMI)	Set of technologies used on the platform.
ComponentLogic (flexible, fixed, rulebased)	Specifies an approach the component logic implementation.
Platform	Defines the component platform. Has several subclasses: ApplicationServer, Hardware, OperatingSystem and VirtualServer
ProgrammingLanguage (Cpp, Java, Ruby, PHP, Erlang, Python, C, C_sharp)	Define programming language used to implement a component.
StatePersistence (Stateless, Statefull)	Specifies whether a component saves internal data during and in between calls of operations on the client's behalf.

TABLE II
PROPERTIES DESCRIBING PLATFORM (SUBCLASSES OF *Platform*).

Property (values)	Description
ApplicationServer	Subclass of Platform. Defines an application server on which a component is deployed, can have such attributes, as: version (string), vendor (string)
JEECompliantAS (TomEE, Glassfish, JBoss, Interstage, JOnAS, Geronimo, SAPNeatWeaver, WebSphere, Resin, ColdFusion, WebLogic)	Subclass of ApplicationServer dedicated to JEE compliant components.
DotNetCompliantAS (AppFabric, IIS, TNAPS, Base4, Mono)	Subclass of ApplicationServer; its individuals define products for .NET enviroment
JavaAS (Jetty, Enhydra, iPlanet)	Application servers for Java environment
Hardware	Subclass of Platform. Used to specify a hardware configuration on which the component is deployed. Attributes: memory (double), processor (string), number_of_cores (int)
OperatingSystem (Windows, Unix, Linux, iOS, Android, Bada, Blackberry)	Subclass of Platform. Defines types of operating systems on which a component is executed. Attributes: version (string), vendor (string), product (string)
VirtualServer (no, yes)	Subclass of Platform. Specifies whether a component is deployed on a virtual server

The ontology provides also a taxonomy of quality attributes. A quality attribute is a nonfunctional characteristic of a component or a system. It represents the degree to which software possesses a desired combination of properties, which are defined by means of externally observable features. Some of the attributes are related to the overall system design, while others are specific to run-time or design time.

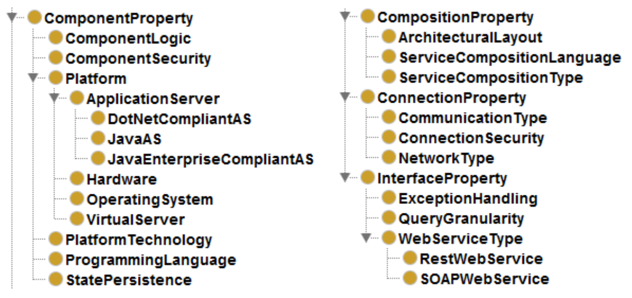

Fig. 3. Classes of properties

SOAROAD ontology defines 30 quality attributes including both terms defined in software quality model by the ISO/IEC 9126-1 norm [9] and those arising directly from requirements to architectures formulated in the SOA manifesto [1]. Examples of classes belonging to the first group are: *Functionality*, *Reliability*, *Usability*, *Efficiency*, *Maintainability* and *Portability*. The example of classes originating from SOA manifesto are *ServiceAutonomy*, *PlatformIndependency*, *LooseCoupling*, *Modularity*, *OpenStandardAdoptation*, *BusinessAgility*, etc.

When designing an applications to meet quality requirements, it is necessary to consider a potential impact of design properties on various quality attributes. SOAROAD ontology defines *influences* object property to this kind of relation.

A design pattern can be seen as a structure build of components of particular types, defining their roles and relations among them together with a set of restrictions on their usage. Design patterns do not change the functionalities of a system but only their organization or structure. One of the most important benefits of using design patterns is that they constitute standardized software building blocks with a well defined influence on quality attributes. In SOAROAD ontology the class *DesignPattern* has 56 subclasses representing patterns dedicated to SOA architecture. The examples of subclasses are: *db.EnterpriseServiceBus*, *db.EventDrivenMessaging*, *db.Orchestration*. The relation *is_described_by* links a particular *CompositionProperty* to one of the defined design patterns.

V. Conclusion

This paper describes the SOAROAD ontology and the concept of a tool supporting documentation of architectures of SOA-based systems. The proposed approach addresses the problem that can be encountered during architecture assessment: to be reliable, a reasoning about architecture qualities, must have solid foundations in a knowledge related to a particular domain: architectural styles, design patterns, used technologies and products. The idea behind SOAROAD ontology is to gather experts knowledge to enable even inexperienced users performing ATAM-based architecture evaluation. An advantage of the presented approach is that its result is a joint representation of architecture views and properties attributed to design elements formalized in OWL language.

From a software engineering perspective, such centralized information resource maintained during the software lifecycle may represent a valuable artifact, which can provide reference to design decisions throughout integration, testing and deployment phases.

On the other hand, a machine interpretable representation, constituting a graph of interconnected objects (individuals), can be processed automatically to check consistency, detect potential flaws and calculate metrics. An extensive list of metrics related to architectural design was defined in [23]. We plan to adapt them to match the structural relations in the SOAROAD ontology, as well to develop new ones.

[1]http://www.soa-manifesto.org/

References

[1] "Archi, archimate modelling tool," 2011, [Online; accessed 23-June-2012]. [Online]. Available: http://archi.cetis.ac.uk/download.html

[2] P. Bianco, R. Kotermanski, and P. Merson, "Evaluating a service-oriented architecture," Carnegie Mellon, Technical Report CMU/SEI-2007-TR-015, September 2007.

[3] N. Bouck'e, D. Weyns, K. Schelfthout, and T. Holvoet, *Applying the ATAM to an Architecture for Decentralized Control of a Transportation System*. Springer, 2006, vol. 4214, pp. 180–198.

[4] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas, "A web-based tool for managing architectural design decisions," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 5, 2006.

[5] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, 2001.

[6] R. C. de Boer, P. Lago, A. Telea, and H. van Vliet, "Ontology-driven visualization of architectural design decisions," in *WICSA/ECSA*. IEEE, 2009, pp. 51–60.

[7] A. Erfanian and F. S. Aliee, "An ontology-driven software architecture evaluation method," in *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*, ser. SHARK '08. New York, NY, USA: ACM, 2008, pp. 79–86.

[8] S. Ferber, P. Heidl, and P. Lutz, *Reviewing product line architectures: Experience report of ATAM in an automotive context*. Springer, 2001, vol. 2290, pp. 364–382.

[9] ISO/IEC, "Software engineering – product quality, ISO/IEC 9126-1," International Organization for Standardization, Tech. Rep., 2001.

[10] L. G. Jones and A. J. Lattanze, "Using the architecture tradeoff analysis method to evaluate a wargame simulation system: A case study," *Technical Report CMUSEI2001TN022 Software Engineering Institute Carnegie Mellon University Pittsburgh PA*, no. December, p. 33, 2001.

[11] Kazman, "Atam:method for architecture evaluation," *CMU-SEI2000TR004*, 2000.

[12] R. Kazman, L. Bass, G. Abowd, and M. Webb, *SAAM: a method for analyzing the properties of software architectures*. IEEE Comput. Soc. Press, 1994, vol. 16pp, no. 5/11/2011, pp. 81–90.

[13] R. Kazman, M. Barbacci, M. Klein, J. Carriere, and S. G. Woods, "Experience with performing architecture tradeoff analysis," *Proceedings of the 21st international conference on Software engineering ICSE 99*, pp. 54–63, 1999.

[14] R. Kazman, L. Bass, and M. Klein, "The essential components of software architecture design and analysis," *Journal of Systems and Software*, vol. 79, no. 8, pp. 1207–1216, 2006.

[15] P. Kruchten, *An ontology of architectural design decisions in software intensive systems*. Citeseer, 2004, pp. 54–61.

[16] J. Lee, S. Kang, H. Chun, B. Park, and C. Lim, "Analysis of VAN-core system architecture- a case study of applying the ATAM," in *Proceedings of the 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, ser. SNPD '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 358–363.

[17] L. Lee and P. Kruchten, *Visualizing Software Architectural Design Decisions*. Springer-Verlag, 2008, vol. 5292, pp. 359–362.

[18] B. Roy and T. C. N. Graham, "Methods for evaluating software architecture : A survey," *Computing*, vol. 545, no. 2008-545, p. 82, 2008.

[19] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[20] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, M. Shaw and D. Garlan, Eds. Prentice Hall, 1996, vol. 123.

[21] P. Szwed, I. Wojnicki, S. Ernst, and A. Glowacz, "Application of new ATAM tools to evaluation of the dynamic map architecture," in *Multimedia Communications, Services and Security*, ser. Communications in Computer and Information Science, A. Dziech and A. Czyżewski, Eds. Springer Berlin Heidelberg, 2013, vol. 368, pp. 248–261.

[22] The Open Group, "Archimate 1.0 specificattion," 2009. [Online]. Available: http://www.opengroup.org

[23] A. Vasconcelos, P. Sousa, and J. Tribolet, "Information system architecture metrics: an enterprise engineering evaluation approach," *The Electronic Journal Information Systems Evaluation*, vol. 10, no. 1, pp. 91–122, 2007.

[24] P. Wallin, J. Froberg, and J. Axelsson, "Making decisions in integration of automotive software and electronics: A method based on ATAM and AHP," *Fourth International Workshop on Software Engineering for Automotive Systems SEAS 07*, pp. 5–5, 2007.